Efficient and Resilient Backbones for Multihop Wireless Networks

Seungjoon Lee Bobby Bhattacharjee Aravind Srinivasan Samir Khuller

***Abstract*— We consider the problem of finding "backbones" in multihop wireless networks. The backbone provides end-to-end connectivity, allowing non-backbone nodes to save energy since they do not have to route non-local data or participate in the routing protocol. Ideally, such a backbone would be small, consist primarily of high capacity nodes, and remain connected even when nodes are mobile or fail. Unfortunately, it is often infeasible to construct a backbone that has all of these properties; e.g., a small optimal backbone is often too sparse to handle node failures or high mobility.**

We present a parameterized backbone construction algorithm that permits explicit tradeoffs between backbone size, resilience to node movement and failure, energy consumption, and path lengths. We prove that our scheme can construct essentially best possible backbones (with respect to energy consumption and backbone size) when the network is relatively static. We generalize our scheme to build more robust structures better suited to networks with higher mobility. We present a distributed protocol based upon our algorithm and show that this protocol builds and maintains a connected backbone in dynamic networks. Finally, we present detailed packet-level simulation results to evaluate and compare our scheme with existing energy-saving techniques. Our results show that, depending on the network environment, our scheme increases network lifetimes by 20–220% without adversely affecting delivery ratio or end-to-end latency.

***Index Terms*— C.2.2 Network Protocols, C.2.8 Mobile Computing, C.2.8.a Algorithm/protocol design and analysis**

I. Introduction

In multihop wireless networks, end-nodes are typically responsible for relaying traffic [1]. However, we often utilize a "connected dominating set" of nodes that form a routing backbone [2, 3]. Nodes not in the backbone have at least one backbone neighbor (hence the backbone is a dominating set). Further, since the non-backbone nodes do not need to route traffic for other nodes, they can save energy by not participating in the routing protocol. If the hardware permits, the non-backbone nodes can further save energy by entering a power-save or sleep mode, only waking up to send messages or check for pending messages. Smaller backbones lead to greater overall energy savings; as such, there have been many algorithms developed for constructing near-optimal connected dominating sets in wireless networks [3–6].[1] The problem of focusing only on the backbone size is that when nodes are battery-powered, the use of low-battery nodes in the backbone can shorten the overall network lifetime. Therefore, many schemes have been proposed that consider the residual battery power in selecting backbone nodes [7–9]. However, along with the battery capacity, these schemes often also use other criteria for including nodes in the backbone (e.g., randomized node selection for arbitration [7]). This leads to the inclusion of low-capacity nodes in the resulting backbone. Clearly, a small backbone composed of high-capacity nodes can significantly increase total network lifetime; the construction of such backbones is the subject of this paper.

The operating environments for multihop wireless networks can vary widely (e.g., minimal node mobility in sensor or rooftop networks [10] vs. higher mobility for rescue operation). Ideally, a backbone construction algorithm should work well in a wide range of network environments. In some existing backbone construction algorithms, nodes use only local information to build and maintain a backbone quickly [2, 7–9]. Although this class of backbone algorithms can be useful in dynamic networks, they do not provide any guarantee on performance objectives such as backbone size or node capacity. Other backbone construction schemes find a "good" connected backbone, e.g., with provable bounds on backbone size or control overhead. However, this second class of algorithms typically have higher control overhead, require longer convergence times, and do not provide efficient mechanisms for backbone maintenance [3, 4]. Therefore, they are most useful in static environments, but in dynamic networks, the overhead of maintaining a "good" backbone can be prohibitive. Due to such inherent heterogeneities in the operating environments for multihop wireless networks, it is unlikely that a single fixed algorithm will work best in all situations. In this work, we develop a general solution that can be tailored to particular network environments.

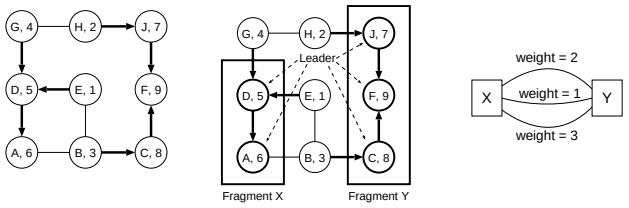
The contributions of this paper are as follows. (i) We present a *parameterized backbone construction algorithm*, which permits explicit tradeoffs between different performance measures including backbone size, resilience to node movement and failure, node capacity, and path lengths. Our scheme has two logical steps. First, each node nominates its highest-capacity neighbor as its "leader." Next, we connect these leaders such that the resulting backbone achieves specific efficiency and resilience properties. (ii) We prove that our scheme can construct essentially *best possible backbones* with respect to node capacity and backbone size. To the best of our knowledge, this is the first work that achieves both objectives at the same time. (iii) Based upon our backbone construction algorithm, we present a *distributed protocol* that builds and maintains a connected backbone in dynamic networks where nodes are mobile, and node capacity constantly changes. (iv) We present *simulation results* that investigate different performance as-

S. Lee is with AT&T Labs, Research (Email: slee@research.att.com). B. Bhattacharjee, A. Srinivasan, and S. Khuller are with the Department of Computer Science, University of Maryland (Email: {bobby,srin,samir}@cs.umd.edu). S. Lee and B. Bhattacharjee were supported in part by NSF Awards CNS-626636, CAREER ANI-0092806, and a fellowship from the Sloan Foundation. A. Srinivasan was supported in part by NSF Award CCR-0208005, NSF ITR Award CNS-0426683, and NSF Award CNS-0626964. Part of his work was done while on sabbatical at the Network Dynamics and Simulation Science Laboratory of the Virginia Bioinformatics Institute, Virginia Tech. S. Khuller was supported in part by NSF grants CCF-0728839 and CCF-0430650.

[1] In general, finding a minimum connected dominating set is NP-hard.

pects of our proposed algorithm, including backbone size, network lifetime, backbone-node capacities, and path lengths. Compared to previous energy-saving techniques, our scheme increases network lifetimes by 20–220% without adversely affecting data delivery or end-to-end latency.

Roadmap. We present the first phase of our algorithm (leader nomination) in § II and the second phase (leader connection) in § III. We present our distributed protocol in § IV and simulation results in § V. We compare our scheme with related work in § VI and conclude in § VII. Please see the Appendix for proofs omitted in the body of the paper.



(a) Leader selection (b) Fragments

Fig. 1. Leader nomination and resulting fragments.

Fig. 2. Multi-graph representation of Figure 1(b).

II. Leader Nomination

In this section, we first describe how each node nominates a *leader* in the initial phase of our algorithm, and then show desirable properties of the resulting set of leaders. We defer the description of connecting the leaders to Section III.

We assume the network is connected and model it as undirected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges between nodes. (We discuss the issue of uni-directional links in Section IV.) We denote the total number of nodes in the network by $n = |V|$. We define $N(v)$ to be the set of neighbors of node v, and $N^+(v) = N(v) \cup \{v\}$. We denote v's degree by $d_v = |N(v)|$, and $\Delta = \max_{v \in V} d_v$. A node v has a unique ID and a capacity value c_v. Although we can consider various attributes for c_v (e.g., CPU speed, storage space), we focus on the battery capacity in this paper.[2]

A. Algorithm Description

We assume that each node knows the capacity value of its neighbors. The algorithm proceeds as follows: each node nominates the node with highest capacity value in $N^+(v)$ as leader. Each node then informs its leader of its decision, and all nominated nodes constitute the set of leaders, which we denote by $\mathcal{L}$. For example, in Figure 1(a), the network has nine nodes. The number in each circle denotes the node capacity (e.g., $c_A = 6$). Thin lines between nodes represent wireless links, while thick lines with arrows represent leader nomination. In the figure, G nominates D because $c_D = 5$ is higher than $c_H = 2$ and $c_G = 4$. As a result, nodes A, C, D, F, and J become leaders, as shown in Figure 1(b). (Nodes A and F nominate themselves as leader, which we do not show here.)

The above algorithm requires only one-hop neighborhood information and constant time. A similar clustering scheme is proposed in [11]. Gao et al. [12] analyze the size of resulting set using specific geometrical properties. However, their analysis assumes that all nodes have a square-shaped communication region of the same size, which is seldom the case in practice [10]. We next present new analysis results, based on more realistic assumptions.

B. Properties of the Leader Set $\mathcal{L}$

We show that $\mathcal{L}$ forms a dominating set using high-capacity nodes, and that the cardinality of $\mathcal{L}$ is small under reasonable assumptions. Recall that a *dominating set DS* of $G = (V, E)$ is a subset of V, where each node in V either is in *DS* or has a neighbor in *DS* [6]. If all nodes in *DS* are connected, then it is called a *connected dominating set (CDS)*. A *minimum* (connected) dominating set is of smallest cardinality among all (connected) dominating sets. We define a *maximum-capacity* (connected) dominating set DS_M to be a (connected) dominating set that maximizes the *bottleneck* node capacity. Formally, DS_M satisfies:

$$\forall\, DS, \min_{v \in DS_M} c_v \geq \min_{u \in DS} c_u, \tag{1}$$

where *DS* denotes a (connected) dominating set.

Theorem 2.1: $\mathcal{L}$ is a maximum-capacity dominating set.

Proof: $\mathcal{L}$ is a dominating set by construction. We prove the maximum-capacity property by contradiction. Assume that $\mathcal{L}$ is not a maximum-capacity dominating set. Consider a maximum-capacity dominating set DS_M. Then, the minimum-capacity node $v \in \mathcal{L}$ satisfies: $\forall u \in DS_M, c_v < c_u$. By the leader nomination rule, there exists a node w for which v is the maximum-capacity node in $N^+(w)$. However, DS_M also has a node u in $N^+(w)$, and since v is w's maximum-capacity neighbor, $c_u \leq c_v$, which is a contradiction. ■

We now show that the expected size of $\mathcal{L}$ (denoted by $E[|\mathcal{L}|]$) is small. For the sake of simpler analysis, we first consider the case of D-regular graphs (i.e., $\forall v,\ d_v = D$) and analyze a more generalized case later in this section. In this analysis, we assume c_v is uniformly distributed between 0 and 1, and $\log(\cdot)$ denotes the natural logarithm. (We plan to extend our analysis for other distributions in the future.)

Theorem 2.2: Suppose $\forall v, d_v = D$ for a positive integer D. Then, there exists a parameter $\epsilon > 0$ such that:

$$E[|\mathcal{L}|] \leq (1 + \epsilon)\tfrac{n}{D} \log(D + 1). \tag{2}$$

Also, ϵ can be arbitrarily close to 0 as D increases.

In practice, wireless nodes are likely to have different numbers of neighbors, and Theorem 2.2 does not hold in general. However, due to spatial locality in the node distribution, we expect that neighboring nodes in multihop wireless networks have a similar number of neighbors. Formally, for a constant $\alpha \geq 1$, we define $G = (V, E)$ to be *α-locally-regular* if $\forall (u, v) \in E,\ d_v \leq \alpha\, d_u$. In a 3-locally-regular graph, for example, the degree of v's neighbor is between $d_v/3$ and $3\, d_v$.

[2]For the ease of exposition, we assume distinct capacity values throughout this paper. In practice, we use unique IDs to break ties.

We now generalize Theorem 2.2 to show that in α-locally-regular graphs, $E[|\mathcal{L}|]$ is within an $O(\log \Delta)$-factor of the size of a minimum dominating set.

Lemma 2.3: Suppose $G = (V, E)$ is α-locally-regular for constant $\alpha \geq 1$. Then,

$$E[|\mathcal{L}|] \leq c' \sum_{v \in V} \frac{1}{d_v} \log (d_v + 1),$$

where c' is a constant that depends on α.

Theorem 2.4: Suppose $G = (V, E)$ is α-locally-regular for constant $\alpha \geq 1$. Then, $E[|\mathcal{L}|] = O(\log \Delta)\, \mathit{OPT}$, where *OPT* is the size of a minimum dominating set.

Theorems 2.2 and 2.4 are essentially best possible. Theorem 2.2 holds for any *D-regular* graph, and as shown in [13], there exist D-regular graphs whose minimum dominating sets are of size at least $(1 - \epsilon')\frac{n}{D} \log D$ for $\epsilon' > 0$. As D becomes large, this value becomes arbitrarily close to the upper bound in Theorem 2.2. Also, the *approximation ratio* of Theorem 2.4 to *OPT* is $O(\log \Delta)$. In general, finding a minimum dominating set for a given graph is NP-hard [6]. Also, no polynomial time algorithm can achieve the approximation ratio of $(1-\epsilon') \log \Delta$ for any $\epsilon' > 0$ unless NP has $n^{O(\log \log n)}$-time deterministic algorithms [14]. Thus, the bound in Theorem 2.4 is within a constant factor of the best-possible approximation. Although other existing schemes provide certain theoretical guarantees about resulting dominating sets, some rely on sequential operation on an entire network or multiple rounds of local operation [3, 4, 15], which can delay the dominating-set formation in distributed environments. Some other analytical results are based on certain simplified assumptions [4, 12]. We use only one round of local operation and provide an analysis for generalized settings that relate to realistic distributed environments.

III. Connecting the Leaders

We now present the second phase of our algorithm that connects the leaders to form a connected dominating set. We describe a special case of our scheme and then present the general scheme (called TRUNC-K, where K is a system parameter).

A. Multigraph Representation

The set of leaders form a forest in which edges are leader-nomination relations. We refer to each tree in this forest as a *fragment*. For example, in Figure 1(b), there are two fragments: fragment X (nodes A and D) and fragment Y (nodes C, F, and J). Since $\mathcal{L}$ is a dominating set, as shown in [6], chains of up to two non-leader nodes are sufficient to connect all fragments. We define a *virtual edge* to be such a chain of (up to two) non-leader nodes that connects two fragments. We transform the graph into a multigraph, where each fragment corresponds to a vertex with (possibly multiple) virtual edges connecting fragments. For a given virtual edge, we use the minimum node capacity as the weight of the edge.[3]

[3]It is possible that no nodes are involved in a virtual edge. In this case, we set the weight of the virtual edge to ∞.

In Figure 1(b), there are three virtual edges between fragments X and Y: the first one using G and H, the second one using E and B, and the last one using B only. The weights of these three virtual edges are 2, 1, and 3, respectively. Figure 2 shows the corresponding multigraph representation. We next describe how we use this multigraph to find a connected backbone.

B. TRUNC-∞: Spanning Tree-Based Algorithm

We begin with an approach based on the well-studied minimum spanning tree (MST) problem. This MST-based approach is a special case of our parameterized algorithm (Section III-C), which we call TRUNC-∞. Recall that an MST of edge-weighted graph $G = (V, E)$ connects all nodes in V using a tree $T \subseteq E$, such that the sum of edge weights in T is minimized [16]. In many algorithms that find MSTs, nodes select a minimum outgoing edge that does not result in a cycle [16, 17]. However, since we want to select high-capacity nodes in the backbone, we need to use *maximum-weight* outgoing virtual edges. For example, in Figure 2, when connecting fragments X and Y to obtain high-capacity connected backbone, we should use the virtual edge of weight 3. We further illustrate this using an example graph in Figure 3: here, each node corresponds to a fragment after the leader nomination phase, and each fragment is connected by virtual edges. (We show only the maximum-weight virtual edges between fragments for clarity.) Figure 4 shows the MST (as defined above).

Let $\mathcal{B}_\infty$ denote the connected backbone obtained by using TRUNC-∞. We next show that $\mathcal{B}_\infty$ produces a *small* connected backbone using *high-capacity* nodes.

Theorem 3.1: TRUNC-∞ results in a maximum-capacity connected dominating set.

Lemma 3.2: $|\mathcal{B}_\infty| \leq 3|\mathcal{L}|$, where $\mathcal{L}$ denotes the leader set.

Proof: If $\mathcal{L}$ has f fragments, we need $(f - 1)$ virtual edges for a spanning tree. Since each virtual edge has up to two nodes and $f \leq |\mathcal{L}|$, $|\mathcal{B}_\infty| \leq |\mathcal{L}| + 2(f - 1) \leq 3|\mathcal{L}|$. ■

From the lemma and Theorem 2.4, the next theorem follows.

Theorem 3.3: For α-locally-regular graphs, $E[|\mathcal{B}_\infty|] = O(\log \Delta)$ *OPT*, where *OPT* denotes the size of a minimum connected dominating set.

Discussion: Theorem 3.1 states that $\mathcal{B}_\infty$ includes a node with low capacity only when it is necessary in maintaining connectivity. We show in Theorem 3.3 that for α-locally-regular graphs, $\mathcal{B}_\infty$ is an $O(\log \Delta)$-approximation to a minimum connected dominating set. As discussed in Section II, this is within a constant factor of best possible approximation. However, if a maximum-capacity backbone of even smaller size is desired, we can further reduce the constant factor by using the following optimization. When the $\mathcal{B}_\infty$ is found, we get to know the minimum node capacity in the backbone (say, c_{th}). Then, we apply any distributed CDS algorithm [3, 4] onto a restricted set of nodes v where $c_v \geq c_{th}$. For example, by applying the scheme in [3], the approximation ratio of the resulting maximum-capacity backbone becomes at most $2 \log \Delta + 3$. (See the last part of the Appendix.)

Although TRUNC-∞ backbones achieve our desired goals (i.e., finding a small backbone using high-capacity nodes),

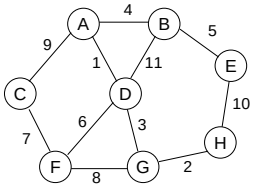


Fig. 3. Example graph.

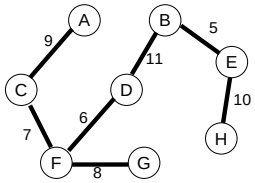
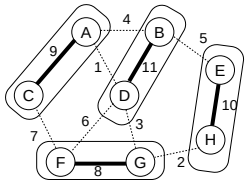
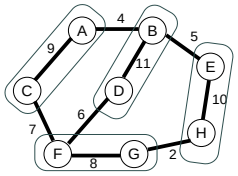


Fig. 4. TRUNC-∞ backbone



(a) After first round



(b) Resulting backbone ($\mathcal{B}_1$)

Fig. 5. Illustration of truncated algorithm.

Algorithm 1 Truncated Algorithm (Centralized)

```
Round ← 0
while more than one fragment exists do
    if Round = K then
        Merge with all neighboring fragments
        Return
    end if
    Each fragment selects the best outgoing edge
    Merge fragments using the selected edges
    Round ← Round + 1
end while
```

the running time can be long. For example, a distributed MST algorithm by Gallager, Humblet, and Spira (the GHS algorithm) takes $O(n \log n)$ running time [17]. Also, there is a clear trade-off between small backbones and shorter path lengths as well as resilience. In Figure 4, the backbone becomes disconnected even when a single link fails. Also, to reach a node in fragment G, a node in fragment H needs to use a path consisting of five virtual edges, compared to only one when no backbone is used. We address this issue next.

C. TRUNC-K: Parameterized Algorithm

We now describe our generalized scheme that balances the above-mentioned trade-off when connecting the leader set (Algorithm 1). It is based on a well-known MST algorithm by Boruvka [18]. In Boruvka's algorithm, each fragment finds and marks the best outgoing edge. Then, using those edges, fragments are merged into new larger fragments. This step is repeated until there is no outgoing edge (i.e., there is only one fragment). During the first K rounds, our algorithm runs just as Boruvka's algorithm, where K is an algorithm parameter. However, in our truncated algorithm, all remaining fragments after K rounds mark edges to *all* neighboring fragments and are merged into one fragment. One extreme case is $K = 0$, where after leader nomination, each pair of neighboring fragments marks one virtual edge (e.g., all edges in Figure 3). Another extreme case is $K = \infty$, which results in $\mathcal{B}_\infty$.

Figure 5 shows the operations of the algorithm applied to the graph in Figure 3. Here, we set $K = 1$. In the first round, each individual fragment selects the best outgoing edge among neighboring fragments, and fragments are merged using selected edges. Then, as shown in Figure 5(a), there remain four fragments at the end of first round. Since $K = 1$ in this example, each remaining fragment after the first round connects to all neighboring fragments. For example, fragment FG chooses three edges to fragments AC, BD, and EH. The resulting connected backbone is shown in Figure 5(b).

We call this algorithm TRUNC-K and the resulting backbone $\mathcal{B}_K$. In contrast to $O(\log n)$ rounds in Boruvka's algorithm, TRUNC-K needs only a constant number (K) of rounds to complete, and the resulting backbone has higher redundancy than $\mathcal{B}_\infty$. This eventually leads to both increased resilience against node mobility and decreased average path length. Note that the resulting backbone is not a maximum-capacity backbone and may include low-capacity nodes. However, by construction, nodes included in the first K rounds are part of a maximum-capacity backbone. After the K-th round, when connecting to each of remaining neighboring fragments, we choose the best virtual edge among typically multiple edges, and we include relatively high-capacity nodes. Also, the resulting backbone includes more virtual edges than $\mathcal{B}_\infty$, and Theorem 3.3 does not hold. However, we can adjust K to control the amount of increase. Our future goal is to analyze the performance trade-offs (e.g., backbone size, capacity distribution) when we vary K. We next use simulation experiments to illustrate that even with small values of K, the increase in backbone size is not significant.

D. Evaluation of the TRUNC-K Algorithm

In this subsection, we use simulations to understand performance trade-offs of the TRUNC-K algorithm (e.g., backbone size, capacity) when we use different values of the parameter K. In this simulation, stationary nodes are distributed on a square uniformly at random, and we vary the number of nodes and the size of square to experiment with various settings. Node capacity values are uniformly distributed between 0 and 1. Nodes within the nominal transmission range (250 meters) become neighbors. For each set of parameters, we use 25 runs with different node placement scenarios and report the average. We also experimented using various scenarios with non-uniform node placement and different capacity distribution, and obtained similar results. (See Section V.)

Backbone Size: In Figure 6, we show the average size of the backbone with varying K. We use two different network settings—the one with 1000 nodes on a 2km×2km square, and the other with 4000 nodes on a 4km×4km square. In Figure 6, the use of extra rounds is most effective when K is small. Specifically, the first round ($K = 1$) leads to the

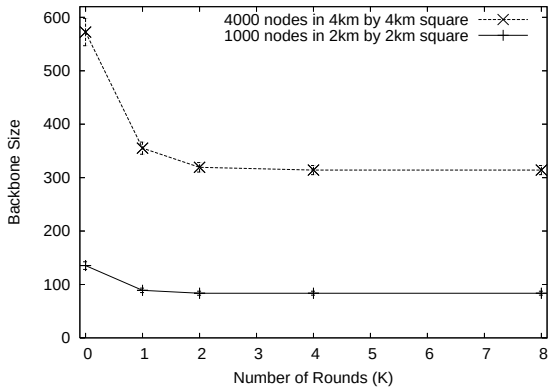


Fig. 6. Backbone size with different K values. The error bars represent standard deviations.

	1.4 km×1.4 km		2.0×2.0		2.8×2.8		4.0×4.0	
	min.	avg.	min.	avg.	min.	avg.	min.	avg.
$\mathcal{B}_0$	0.349	0.913	0.111	0.854	0.021	0.771	0.007	0.666
$\mathcal{B}_1$	**0.860**	0.967	**0.705**	0.934	**0.307**	0.872	**0.054**	0.760
$\mathcal{B}_2$	0.888	0.970	0.787	0.942	0.573	0.892	0.188	0.796
$\mathcal{B}_\infty$	**0.888**	0.970	**0.787**	0.942	**0.574**	0.893	**0.200**	0.802

TABLE I

CAPACITY VALUES OF BACKBONE NODES WITH VARYING NODE DENSITY

largest reduction in backbone size. With larger $K > 2$, there are a small number of remaining fragments after K rounds. As a result, when compared to $\mathcal{B}_\infty$, connecting all neighboring fragments does not significantly increase the backbone size.

Capacity Distribution among Backbone Nodes: We investigate another potential problem with TRUNC-K backbone—the resulting backbone may include low-capacity nodes. In this scenario, we use 1000 nodes but vary the square size, thus varying node density. In Table I, we list the minimum and average capacity values depending for K values with different node density. Even with small K (1 or 2), the difference in minimum-capacity between $\mathcal{B}_K$ and $\mathcal{B}_\infty$ is small. For example, in the case of 2km×2km square, the difference between $\mathcal{B}_1$ and $\mathcal{B}_\infty$ is around 10% (0.705 vs. 0.787). The difference in average capacity values is even smaller: less than 1% for the same scenario. As discussed in III-C, this is because fragments after K rounds choose best possible virtual edges to connect neighboring fragments, and very low-capacity nodes are not likely to join the backbone. However, in sparser networks, fewer virtual edges are available between neighboring fragments, and the difference in minimum capacity between $\mathcal{B}_K$ and $\mathcal{B}_\infty$ is slightly larger.[4] In the actual deployment of the TRUNC-K algorithm, we should choose an appropriate K value based on network parameters such as node density and mobility.

Average Path Length: We consider the average shortest path length (induced by the backbone) between each of all possible node pairs. This measure provides a good lower bound for the performance of practical routing protocols such as [1]. Due to space constraints, we report results for the cases with 1000 nodes placed in a 4km×4km square. When there is no backbone, the average path length is 11.2. The use of any routing backbone inevitably increases path length since we are forced to find paths using a restricted set of nodes. $\mathcal{B}_0$ has most redundancy, and the average path length is 12.5 with minimum increase. As K increases, the redundancy decreases and the path length thus increases; the average path length of $\mathcal{B}_1$ is 14.8, while that of $\mathcal{B}_2$ is 18.8. In contrast, the average expansion in path length for $\mathcal{B}_\infty$ is 23.0, which is more than twice the underlying shortest paths. We observe that small K values again offer a good trade-off. For example, compared to $\mathcal{B}_\infty$, the average path length of $\mathcal{B}_1$ is up to 36% shorter, while the backbone size is only up to 13% larger.

To summarize, backbones obtained using small K (1 or 2) perform well and provide a reasonable balance among a number of performance measures. We next describe a distributed protocol that implements the TRUNC-K algorithm.

[4]When 5km×5km squares are used with 1000 nodes, only four cases out of 25 resulted in connected networks, and the use of 4km×4km squares with 1000 nodes corresponds to considerably sparse scenarios.

IV. DISTRIBUTED PROTOCOL DESCRIPTION

In this section, we present our distributed protocol that implements the TRUNC-K algorithm to construct and maintain a connected backbone in *dynamic* network environments. Our protocol is based on the GHS algorithm, which is a distributed version of Boruvka's algorithm [17]. We assume that each node has a unique ID (e.g., IP address). We first describe the leader nomination and explain how to connect the fragments obtained after the nomination phase. We also present a backbone maintenance mechanism later in this section. These backbone construction operations are performed periodically. While the interval depends on various factors such as mobility, node density, traffic pattern, etc., our simulations in Section V indicate that 10–20 seconds works well in typical scenarios.

A. Leader Nomination Protocol

Each node broadcasts a HELLO message periodically that includes information about itself and its neighbors. Figure 7 shows the fields for individual node information in HELLO messages. Using these fields, each node maintains information about two-hop neighbors (e.g., capacity, fragment root IDs).

Before broadcasting a HELLO message, node v checks which neighbor has the highest capacity (e.g., residual battery power). Suppose u is the highest-capacity neighbor of v. Then, v sets its *Leader ID* field to u in its HELLO message. Upon receiving a HELLO message from v, u becomes a leader and sets its *IsLeader* field to TRUE in subsequent HELLO messages until v changes leaders and there exist no other neighbors nominating u (e.g., due to later decrease in residual battery).

Suppose node u finds itself as the highest-capacity node in $N^+(u)$. Then, in addition to being a leader, u also becomes a *level-0 fragment root*, where a level-0 fragment is a set of leaders who are themselves connected via the leader-nomination relation. In Figure 1(b), nodes A and F are level-0 fragment roots.

B. Protocol for Fragment Members

As discussed in Section III, the set of leaders form a forest consisting of multiple fragments, and the protocol described

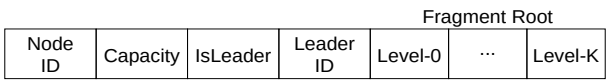


Fig. 7. Information about individual nodes in a HELLO message.

here merges the fragments to form one connected component. In Algorithm 2, we present a high-level protocol description. We begin with the operation of level-0 fragments and later discuss the operation of higher-level fragments.

We illustrate the protocol operations of level-0 fragments using Figure 8. Each level-0 fragment forms a tree rooted at its fragment root. To discover neighboring level-0 fragments, level-0 fragment roots periodically send REQ_0 messages, which are forwarded down this tree to the leaves (who cannot forward the REQ message any further). The leaves then generate a REPLY_0 message that contains information about other fragments (if any) that they are connected to. The REPLY_0 messages are forwarded back towards the fragment root. For example, in Figure 8(b), node D generates a REPLY_0 message, which contains the ID of other level-0 fragments that D knows of (F_y in this example) along with the cost of the virtual edge to connect to F_y. (Recall that D keeps the information about fragment roots of two-hop neighbors.) At each hop, before forwarding the REPLY message towards the leader, nodes update the message if they know of a better virtual edge than the one carried in the message. Also, nodes add information about any new neighboring fragments that are not in the REPLY message. In our example, node B does not modify the REPLY message from D, since its path to F_y is worse than the one that D found (Figure 8(b)). (A and C also send REPLY_0 messages, which are not shown in the figure.)

Once the fragment root has accumulated all REPLY messages (or has timed out on some), it sends a CONNECT message using the best outgoing virtual edge. This is shown in Figure 8(c), where X connects to Y using the weight 7 edge through D. This virtual edge has two non-backbone nodes P and Q, and upon receiving the CONNECT_0 message, they become *bridges* and join the backbone. Using these bridge nodes, F_x and F_y are merged to form a new level-1 fragment.

Level-1 fragments also need to find neighboring level-1 fragments to form next-level fragments. To elect level-1 fragment roots that send REQ_1 messages, we use the following rule similar to [17]: If two fragments choose each other as their best neighboring fragment, then two fragment roots become candidates for the next-level fragment root. We choose the node with lower ID as the level-1 fragment root.

Level-i fragments ($0 < i < K$) operate similarly to above procedures until their level reaches K. At the highest level-K, instead of using only the best virtual edge, the level-K fragment roots send CONNECT messages to *all* neighboring level-K fragments, thus assuring a connected backbone.

In Figure 8, X is a both level-0 and level-1 fragment root, and it periodically sends both REQ_0 and REQ_1 messages. In general, a node can be a fragment root of up to (K+1) levels at the same time. Even if higher-level fragments are already found, lower-level fragment roots (e.g., Y in Figure 8) still send REQ_i messages periodically. This allows lower-level fragments to find new or better virtual edges to neighboring fragments in dynamic networks.

Algorithm 2 Distributed operation of level-i fragments

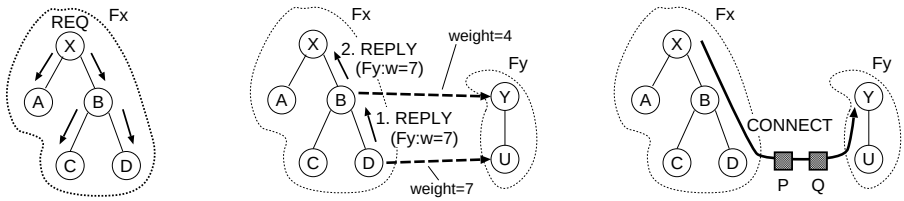
1: Level-i fragment root periodically sends REQ_i message
2: Fragment members send REPLY_i messages with neighboring level-i fragment information
3: **if** level-i fragment root receives all REPLY_i messages, or a timeout occurs **then**
4: **if** i = K **then** {highest level}
5: Fragment root sends CONNECT_i messages to all neighboring level-i fragments
6: **else** { $i < K$}
7: Fragment root sends CONNECT_i message only to best neighboring level-i fragment
8: **end if**
9: **end if**

C. Backbone Maintenance

All of the protocol specific states (e.g., leaders, bridges, fragment roots) are "soft." A node removes a neighbor if it does not receive a HELLO message from the neighbor for a certain duration (e.g., four HELLO-PERIODs). If the capacity of the leader becomes lower (e.g., due to battery consumption), a node may choose a different node with highest capacity as leader. If a leader finds that no neighbors nominate it as leader for some time, it stops being a leader. When a bridge does not receive a CONNECT message for a certain period of time, it stops being a bridge.

In a dynamic network, however, the basic protocol mechanisms described above may not be sufficient for the timely maintenance of the connected backbone. We efficiently reconstruct the backbone using a simple local search protocol that exploits spatial locality. We illustrate its operation via an example. In Figure 9, node P detects a link failure to backbone neighbor Q. P looks up its neighbor table to find other nodes that also had Q as neighbor. (Note that these nodes need not currently be part of the backbone). In this example, P finds two such neighbors, M and N, and sends a RECOVER message to N, which has higher capacity. Upon receiving this message, N temporarily joins the backbone and forms a bridge to Q. In the next REQ-REPLY phase, X might choose a different virtual edge (of higher weight) to connect to F_y. If that happens, N will leave the backbone after a timeout.

There are potential problems with the local recovery scheme. First, the repaired backbone may include lower-capacity nodes than necessary. However, as mentioned above, in the next REQ-REPLY phase, the fragment root will discover the best virtual edge and send the appropriate CONNECT message. Also, a node may not be able to find a common neighbor for recovery. However, in networks with reasonable node density, such events will likely be infrequent. Finally, the recovery scheme does not help when nodes fail. However, the TRUNC-K backbone should have sufficient resilience to maintain connectivity against infrequent recovery failures.



(a) REQ message (b) REPLY message (c) CONNECT message

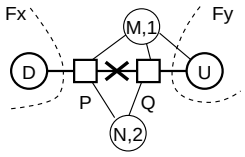


Fig. 9. Local maintenance.

Fig. 8. Overview of protocol operations. In (b), REPLY message contains information about outgoing virtual edges.

We examine the effectiveness of this recovery scheme using simulations in Section V.

Discussion: As shown in Figure 7, HELLO messages in TRUNC-K contain (K+3) node IDs per neighbor: *Node ID, Leader ID*, and *fragment roots* for $K+1$ levels. For example, suppose that node U in Figure 8(c) is about to broadcast a HELLO message. When K=1, the information for neighbor Q should include (Q, U, Y, X). Including more information in HELLO messages increases the control overhead. However, since many neighbors share leaders and fragment roots, we can reduce the increase by using the following simple indexing scheme. U arranges all neighbors into an array and uses the index values for leader IDs and fragment roots of its neighbors. In the above example, U includes the following information for its neighbor Q: (Q, I_U, I_Y, I_X). Since an index field is typically shorter (e.g., 1 byte) than an actual ID (e.g., 4 bytes), the amount of length increase becomes smaller. Note that U still needs to include some non-neighbor IDs (e.g., X) in its HELLO messages. However, since many of its neighbors share leaders and fragment roots, the number of such non-neighbors nodes are likely to be small. TRUNC-K also uses a few additional control messages (e.g., REQ, REPLY), which can potentially increase the overall control overhead. Our simulation results in Section V-C show that the overall control overhead of TRUNC-K is minimal.

Wireless links in practice can be uni-directional and show a wide range of difference in their quality [10]. In our protocol, we can easily detect uni-directional links using the neighbor information in periodic HELLO messages and exclude those links when building backbones. We further discuss the issue of link quality in Section V.

V. SIMULATION STUDY

In this section, we compare TRUNC-K with prior approaches using simulation experiments. Based on the results in Section III-D, we consider only the case of $K = 1$. Although we performed experiments in various other scenarios using different topologies, traffic patterns, and capacity distributions, due to space constraints, we present only a subset of representative results in this paper. While we focus mainly on prior schemes that consider node capacity, at the end of Section V-B, we also compare TRUNC-K with other existing schemes that do not consider node capacity. We first describe prior capacity-aware approaches and then compare the performance of our scheme against them.

A. Brief Description of Existing Schemes

In SPAN [7], a node becomes a *coordinator* and joins the backbone when any two neighbors are not connected using up to two current coordinators. To minimize contention and give priority to high-energy nodes, SPAN uses a randomized backoff using the energy level, number of neighbors, and a random number. A coordinator withdraws after some period of time to give other neighbors a chance to become coordinators.

In GAF [8], the area is divided into square-shaped virtual grids. GAF assumes the availability of location information (e.g., from GPS), and each node can know its virtual grid from the location information. Then, GAF elects the highest-energy node in each grid, and these elected nodes form a connected backbone due to the grid construction rule [8].

In the scheme by Wu et al. [9], a node initially joins the backbone if its two neighbors are not connected. Then, to reduce the size of this initial backbone, node v searches for a neighbor u, or two neighbors u and w, such that the (union of) neighbor set(s) includes the neighbor set of v. Due to symmetry, the above rule may lead to connectivity loss, and the authors of [9] also use the power level and degree of node.

B. Comparison Study in Large Networks

In this set of experiments, we use the same settings as in Section III-D. We measure the performance when the initial backbone stabilizes, and report the average of 25 runs each.

We first examine the size of backbones constructed by different schemes. In this set of experiments, we vary the number of nodes and the size of square, but maintain the average node degree constant. In Table II, we present the average backbone sizes for various scenarios. The standard deviations are small (less than 6% of the average in all cases), which we do not present here. We observe that TRUNC-1 backbones are smallest in all cases. Specifically, when the network has 4000 nodes, the TRUNC-1 backbone has 355 nodes on average. This is 24% smaller than the SPAN backbone, which is the second smallest in all these experiments.

Our proposed scheme also builds a backbone consisting of higher-capacity nodes. In Table III, we tabulate the minimum and average capacity values of backbone nodes. In all cases, the backbone by TRUNC-1 achieves the highest values for both minimum and average node capacity. For example, in 4000-node networks, the TRUNC-1 backbone does not include any of bottom 30% nodes, while the GAF backbone includes some of

	Number of nodes (square size in km×km)			
	500 (1.4×1.4)	1000 (2.0×2.0)	2000 (2.8×2.8)	4000 (4.0×4.0)
SPAN	54.5	113.9	227.7	467.4
Wu et al.	67.4	150.3	308.9	652.5
GAF	158.0	308.6	605.6	1236.9
TRUNC-1	**44.7**	**89.0**	**174.6**	**355.1**

TABLE II

BACKBONE SIZE BY DIFFERENT SCHEMES.

	No. of Nodes (square size in km×km)							
	500 (1.4×1.4)		1000 (2.0×2.0)		2000 (2.8×2.8)		4000 (4.0.×4.0)	
	min	avg	min	avg	min	avg	min	avg
SPAN	0.056	0.700	0.046	0.686	0.025	0.704	0.011	0.708
Wu et al.	0.005	0.504	0.002	0.480	0.002	0.495	0.002	0.504
GAF	0.032	0.714	0.014	0.707	0.007	0.720	0.003	0.723
TRUNC-1	**0.752**	**0.937**	**0.705**	**0.934**	**0.502**	**0.933**	**0.335**	**0.933**

TABLE III

MINIMUM AND AVERAGE CAPACITY VALUES OF BACKBONE NODES

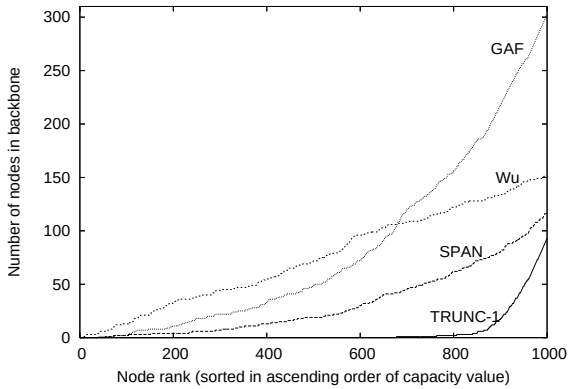


Fig. 10. Capacity distribution of backbone nodes in different schemes.

bottom 0.5% nodes. In the same scenario, the average capacity of TRUNC-1 backbone is also 30% higher than those of SPAN and GAF. When the routing backbone is used to reduce power consumption and increase the network lifetime, the use of low-capacity nodes can drain their energy unnecessarily. We later investigate this aspect using packet-level simulations.

In Figure 10, we present a detailed snapshot of a representative run with 1000 nodes. We sort all nodes in an ascending order of capacity value and cumulatively plot the number of backbone nodes whose capacity is less than or equal to that of a given node. For example, the GAF backbone includes 49 nodes out of 500 lowest-capacity nodes, while SPAN chooses 19 nodes from the lowest 500 nodes. In contrast, the TRUNC-1 backbone does not include any of the lowest-capacity nodes, but selects only 93 nodes among the top 330 nodes.

In Table IV, we report the average path lengths by different schemes as well as the case using no backbones. Not surprisingly, since TRUNC-1 backbones are smaller in size than any other scheme (Table II), its average path lengths are the longest. However, the amount of reduction in backbone size is more than the increase in the path length, especially in larger networks. Specifically, in 4000-node networks, the difference in the average path length between SPAN and TRUNC-1 is around 20%, while the difference in backbone size is more than 30%. The other two schemes (GAF and Wu et al.) have shorter path lengths on average, but their backbones are substantially larger in size (Table II). This result illustrates that TRUNC-1 backbones provide relatively good paths considering the small size.

Experiments with Non-uniform Node Distribution: In the previous experiments, we placed nodes uniformly at random. In this set of experiments, we first partition the area into square grids (with each side 200m) and let each cell potentially have a widely varying number of nodes. Specifically, we randomly choose the node count for each cell using two different distributions: exponential and Pareto. (Within a single cell, nodes are randomly distributed.) In Table V, we report the results when we use a 2km-by-2km area and the average of 10 nodes in each cell. We observe that TRUNC-1 consistently outperforms existing schemes when nodes follow different distribution patterns.

Comparison with Capacity-unaware Schemes: While we have compared TRUNC-1 with prior backbone construction schemes that consider node capacity, there are other schemes that focus on small backbone size or control overhead without considering backbone capacity [3, 4, 19]. Basagni et al. [20] compare some of them based on multiple criteria, and we briefly compare them with TRUNC-1 in terms of backbone size. When we use the same network settings, TRUNC-1 finds smaller backbones than highly localized schemes (e.g., [19]), while TRUNC-1 backbones are similar in size to those based on global operation (e.g., [4]). For example, in dense networks (with the average node degree being around 20), both TRUNC-1 and the scheme by Wan et al. [4] include around 19% of nodes in the backbone on average, while backbones by Dai and Wu [19] include around 25% of nodes. This is somewhat expected; TRUNC-1 uses information about extended neighborhood (e.g., level-1 fragments) and thus can build smaller backbones than purely localized schemes, while the difference from the globally sequential scheme by Wan et al. is minimal (See Section III-C). In addition to small size, TRUNC-1 considers node capacity, and the resulting backbones consist mostly of high-capacity nodes. We next present packet-level simulation results to demonstrate that TRUNC-1 leads to significant increase in network lifetime, while maintaining connectivity in dynamic scenarios.

C. Packet-level Simulations

In this subsection, we focus on saving energy and extending network lifetime using ns-2 simulations [21]. SPAN and TRUNC-1 performed best in Section V-B, and we compare only these two schemes here. We use the SPAN simulation code written by the authors of SPAN.[5] Due to high resource requirements, we have been able to perform simulations only

[5]Available at http://www.pdos.lcs.mit.edu/span/. We plan to make our simulation code public as well.

	No. of nodes (square size in km×km)			
	500 (1.4×1.4)	1000 (2.0×2.0)	2000 (2.8×2.8)	4000 (4.0.×4.0)
No backbone	3.66	5.04	6.86	9.60
SPAN	4.39	6.17	8.44	11.89
Wu et al.	4.14	5.75	7.86	11.01
GAF	3.93	5.45	7.45	10.46
TRUNC-1	5.66	7.84	10.27	14.35

TABLE IV

AVERAGE PATH LENGTH BY DIFFERENT SCHEMES.

	Exponential Distribution			Pareto Distribution		
	backbone size	avg. capacity	min. capacity	backbone size	avg. capacity	min. capacity
SPAN	104.16	0.660	0.017	111.48	0.669	0.025
Wu et al.	118.28	0.494	0.006	131.32	0.507	0.006
GAF	272.56	0.699	0.009	298.12	0.689	0.009
TRUNC-1	87.76	0.919	0.563	89.84	0.923	0.600

TABLE V

BACKBONE SIZE AND CAPACITY VALUES WITH DIFFERENT NODE DISTRIBUTIONS

with relatively small topologies (with 150 nodes). We first describe our simulation environment before reporting the results.

1) Simulation Environment: Both TRUNC-K and SPAN run on top of the IEEE 802.11 MAC layer [22], and non-backbone nodes stay in the power saving mode. In the IEEE 802.11 power saving mode, time is partitioned into *beacon periods*. All nodes stay awake in the beginning of each beacon period and exchange messages to inform neighbors of pending messages. If a node finds that there are buffered incoming messages, it requests the messages and stays awake during the beacon period. Otherwise, it goes back to sleep until the start of the next beacon period. Power saving mode usually leads to increased delay and reduced throughput (e.g., due to additional control packets), and Chen et al. [7] slightly modified the power saving mode in the 802.11 MAC to improve performance, which we use in our simulations.

We assume there are three classes for the node energy level. A low-energy node has 300J of energy, which is used in the experiments in [7]. A medium-energy node has 600J, and a high-energy node has 2500J. (2500J is usually sufficient to last 3000 seconds of simulation time.) We vary the node percentage of each class to examine the performance in different settings. Node energy is constantly updated using the following power consumption values reported in [7]: 1.4W for transmission, 1.0W for receiving, 0.83W for idling, and 0.13W for sleeping. We place 150 nodes uniformly at random on a 1000 meter by 1000 meter square area. The transmission range of each mobile node is 250 meters.

We choose 10 pairs of source and destination nodes uniformly at random among high-energy nodes; each source generates traffic 50 seconds into the simulation at the constant rate of one 128-byte packet per second. The MAC-level transmission rate is 2 Mbps. As we discuss later, SPAN rotates backbone nodes frequently (e.g., 2 changes per second), which shortens path lifetimes. When we used on-demand routing protocols over SPAN, the path maintenance overhead was high. Instead, we use an idealized scheme for packet routing, where a path is found on top of the connected backbone using the centralized Floyd-Warshall algorithm [16] implemented in ns-2. This corresponds to a best case scenario for SPAN. Nodes move according to the Random Waypoint mobility model (pause time=400s, and maximum speed is 1–16m/s) [21]. We also set the minimum speed to be 0.1m/s to avoid speed decay [23]. Unless otherwise stated, we use mobile scenarios with the maximum speed of 1m/s.

In both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. For TRUNC-1, we set the period of REQ messages to 14 seconds, which leads to reasonable performance. In each case, we report the average of 5 runs.

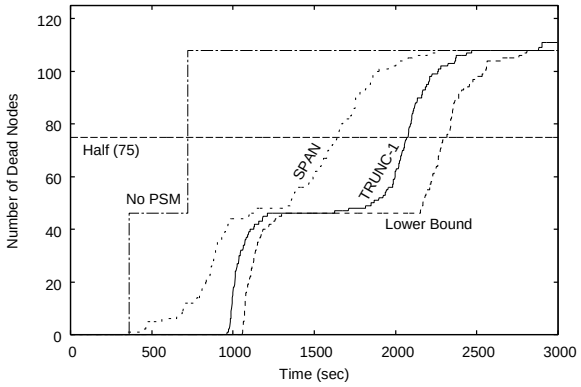


Fig. 11. Number of dead nodes over time. L:M:H=3:4:3.

2) Simulation Results: For the first set of results, we examine two types of network lifetimes [24]. Network *1-life* is the time when the first node dies, and *half-life* is the time when the half of initial nodes die.[6] In addition to TRUNC-1 and SPAN, we use two other schemes for reference. The first one is to identify a lower bound, in which all nodes always stay in sleep mode except when they wake up at the beginning of beacon periods. Each node also sends a 128-byte HELLO message every two seconds. In the second scheme (No-PSM), no power saving operation is used, and all nodes always stay awake without sending any control messages. We send no data traffic in either of the two reference cases.

In Figure 11 we present a snapshot for the number of dead nodes over time. In this setting, approximately 30% of nodes are low-energy (L), 40% of them are medium-energy (M), and the rest 30% are high-energy (H) nodes. (To denote this ratio, we use an abbreviated notation L:M:H=3:4:3.) In Figure 11, the network 1-life of SPAN is similar to that of "No-PSM." This is expected from Figure 10 to some extent; SPAN includes low-energy nodes in the backbone, and their lifetimes decrease significantly. Although SPAN rotates the backbone node responsibility, there exists an unfortunate low-energy node in most of our experiments that stays in the backbone during the first 350 seconds. In contrast, with TRUNC-1, the network 1-life is close to 960 seconds, which is 2.7 times longer than that of SPAN. This is because the TRUNC-1 backbone consists mostly of high-energy nodes plus a few medium-energy nodes, and low-energy nodes can stay in sleep

[6]We assume that the network needs external support after this time (e.g., addition of fresh nodes in the case of sensor networks).

Ratio of L:M:H	1-Life (sec) TRUNC-1	1-Life (sec) SPAN	Half-Life (sec) TRUNC-1	Half-Life (sec) SPAN
4:4:2	892.1 (101.0)	365.5 (28.3)	1911.0 (139.6)	1506.3 (54.5)
3:4:3	946.6 (22.0)	375.0 (29.1)	2106.2 (33.7)	1689.8 (40.5)
2:4:4	962.0 (13.7)	412.3 (54.7)	2208.3 (41.1)	1842.3 (43.9)

TABLE VI

NETWORK LIFETIMES WITH VARIED PROPORTION OF NODES AT DIFFERENT ENERGY LEVELS. THE VALUES IN PARENTHESES ARE STANDARD DEVIATIONS.

	Energy consumption ratio (Idle:Sleep) 6.3:1 [7]	14.0:1 [25]	40.0:1 [8]
1-life	152	189	220
half-life	24	34	43

TABLE VII

LIFETIME EXTENSION IN DIFFERENT ENVIRONMENTS.

	1 packet/sec	2 packets/sec	4 packets/sec
SPAN	0.96 (0.04)	0.88 (0.05)	0.62 (0.02)
TRUNC-1	0.98 (0.02)	0.92 (0.05)	0.58 (0.04)

TABLE VIII

AVERAGE DELIVERY RATIO WITH VARYING TRAFFIC LOAD.

mode and save energy. In the case of TRUNC-1, we observe a sharp increase in the number of dead nodes as the first node dies. This is the time (960 seconds) when all low-energy nodes in TRUNC-1 run out of power. Note that this is earlier than the case of lower-bound (around 1050 seconds). This is because with TRUNC-1, nodes consume more energy to exchange larger HELLO messages (in this experiment around 211 bytes on average) than the lower-bound case (128 bytes). Compared to SPAN, TRUNC-1 also increases the average lifetime of low-energy nodes by 28% (1038.1 seconds vs. 811.3 seconds).

We now consider the lifetime of medium-energy nodes in Figure 11. The use of low-energy backbone nodes in SPAN allows more medium-energy nodes to be in sleep mode and increase their lifetime. Still, compared to SPAN, the TRUNC-1 backbone increases the network half-life by around 26%. We explain this as follows. In this network setting, a connected backbone needs to use several medium-energy nodes to maintain connectivity. Ideally, as the initial medium-energy backbone nodes expend their energy, they should be replaced with different medium-energy nodes, such that their lifetimes do not decrease significantly. From Figure 11, we infer that TRUNC-1 evenly distributes the backbone responsibility among all medium-energy nodes, and no medium-energy nodes die until 1600 seconds. (In Figure 11, after all low-energy nodes die in the case of TRUNC-1 backbone, we observe a relatively stable period during which no node dies.) In contrast, in SPAN, medium-energy nodes start to die before 1200 seconds, and the network half-life of SPAN decreased.

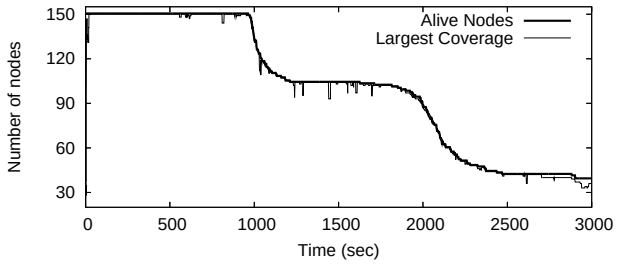
In Table VI, we tabulate the average network lifetimes and standard deviations while varying the proportion of nodes at different energy levels. We observe that in all scenarios, TRUNC-1 achieves longer network lifetimes than SPAN (133–152% longer for 1-life and 20–26% longer for half-life). We also experimented using different parameters (e.g., different initial battery capacity values and L:M:H ratios), and TRUNC-1 outperformed SPAN in all cases. In all these experiments, the average backbone sizes of TRUNC-1 and SPAN are very similar (between 21 and 23 nodes depending on the scenarios).

In the previous experiments, we have used the energy consumption values reported in [7]. In Table VII, we briefly compare the results when we use different sets of values reported in [25] and [8]. We only show the ratio between idle and sleep energy consumption, which is the most dominant difference between these sets. Compared to the results in Table VI, as nodes in sleep mode consume less energy, TRUNC-1 achieves larger network lifetime extension over SPAN (e.g., up to 220% increase in 1-lifetime, compared to previous 152%).

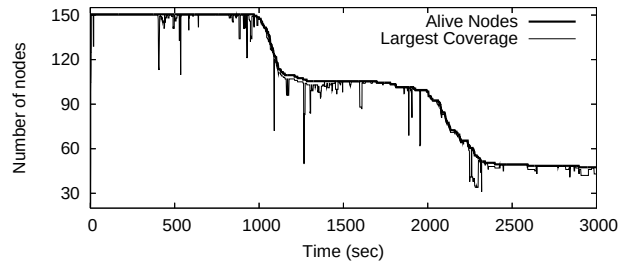
Backbone Maintenance: We now examine the backbone resilience as well as the effectiveness of our proposed maintenance mechanisms against backbone partition. Let us define the *coverage* of a connected backbone to be the number of nodes that are in the backbone or have a neighbor in the backbone. For an ideal connected backbone, its coverage would always be equal to the number of nodes alive in the network. In Figure 12, we show the *largest* coverage of the TRUNC-1 backbone over time, while we change the maximum speed (1m/s, 8m/s, and 16m/s). Due to node mobility or energy-level change, the backbone may get disconnected, and we see occasional drops in the coverage of TRUNC-1 backbone. However, our protocol detects such disconnections quickly, and the local maintenance scheme helps to regain the perfect coverage in a short period of time. As node mobility becomes higher, we observe modest increase in the number of partitions in the TRUNC-1 backbone. Compared to the TRUNC-1 backbone, the SPAN backbone results in more frequent coverage loss (Figure 13). In SPAN, nodes periodically leave the backbone only after ensuring that the departure does not cause backbone disconnection. However, it is possible that a node makes such a decision based on outdated information (e.g., due to node mobility), which occurs frequently, for example, once every 30 seconds on average in Figure 13(a).

Another aspect of backbone maintenance is the frequency with which nodes in the backbone change. In Figure 13(a), the SPAN backbone undergoes 674 membership changes between 100 and 400 seconds. This is because backbone nodes in SPAN periodically leave the backbone. In the same scenario, TRUNC-1 causes 49 changes in the backbone membership. Suppose that an on-demand routing protocol such as DSR [1] found a path using backbone nodes. With SPAN, nodes on such a path are likely to leave the backbone about 12 times more frequently than TRUNC-1, and the source may need to find a new path consisting of backbone nodes frequently. (Recall that this is why we use the idealized routing in our experiments.)

Data Delivery: We briefly report the results about data delivery performance of TRUNC-1 backbone. In the previous light-load experiments, both TRUNC-1 and SPAN achieve near-perfect data delivery ratios. In this set of experiments we experiment with high-load scenarios using 1024-byte data packets with varying packet rates. We also use static networks and ensure the distance between source and destination is more

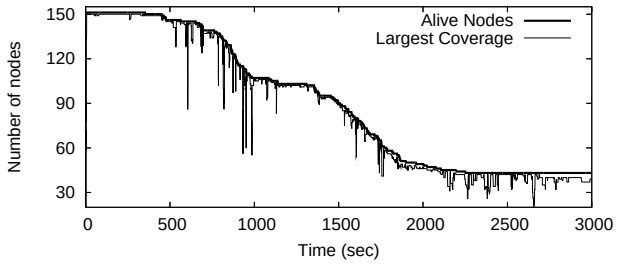


(a) Maximum speed=1m/s. (TRUNC-1)

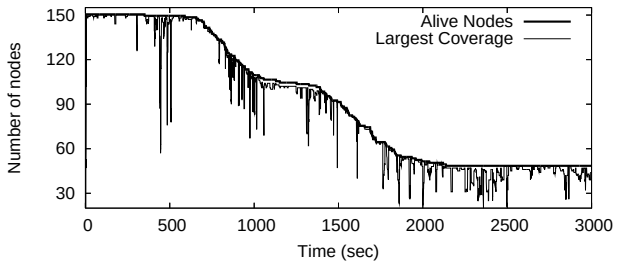


(b) Maximum speed=16m/s. (TRUNC-1)

Fig. 12. TRUNC-1 backbone coverage.



(a) Maximum speed=1m/s. (SPAN)



(b) Maximum speed=16m/s. (SPAN)

Fig. 13. SPAN backbone coverage.

Max. speed	REQ	REPLY	CONNECT	RECOVER	HELLO
1m/s	1.80	2.15	1.64	0.12	52.73
8m/s	2.02	2.29	1.94	0.34	53.43
16m/s	1.83	2.04	2.14	0.49	53.65

TABLE IX
CONTROL PACKETS PER SECOND IN ENTIRE 150-NODE NETWORK.

than 500 meters such that all data packets go through at least two intermediate hops. In Table VIII, we tabulate the average data delivery ratios and standard deviations (in parentheses) with different sending rates. As the amount of data traffic increases, the average delivery ratio decreases in both schemes, and the difference between TRUNC-1 and SPAN is marginal. In these experiments, TRUNC-1 leads to shorter average end-to-end delays than SPAN, but the difference is not significant.

Control Overhead: In both TRUNC-1 and SPAN, each node sends a HELLO message every two seconds. HELLO messages in TRUNC-1 contain more information, and the average message is longer than that of SPAN. Specifically, in TRUNC-1, the average length of HELLO messages is around 192 bytes, and in SPAN it is around 131 bytes. Note that the difference is due in part to more dead nodes in SPAN, which lead to fewer neighbors in HELLO messages.

TRUNC-1 uses additional control messages (e.g., REQ and CONNECT). In Table IX we tabulate the average numbers of control packets per second used in the *entire* network. In TRUNC-1, the total number of non-HELLO control packets is only around 6 packets per second in the 150-node network, and their average sizes are 20 to 70 bytes. While the expected number of HELLO messages is 75 per second (with 150 nodes sending once every two seconds), due to dead nodes, the number in the table is around 30% smaller. We also observe that the overall increase in control overhead due to higher mobility is marginal. We believe that the advantages of TRUNC-1 (e.g., longer network lifetime, better backbone coverage) outweigh the modest increase in control overhead.

Experiments in Lossy Environments: Wireless links are more prone to frame errors than wired links, and a large amount of research attempts to improve system performance in this environment [10, 26]. We briefly report the results when TRUNC-K operates in such lossy wireless environments. In our experiments, we randomly select 20% of links, which intermittently experience a high frame-error rate in which 80% of frames are lost. We vary the length of high-error period and experiment with two scenarios: links experience average of (1) 10-second lossy period in every 100 seconds and (2) 30-second lossy period in every 100 seconds. Our results show that without any loss, the delivery ratio is 99.2%, while it goes down to 96.9% and 91.4% with 10-second and 30-second lossy periods, respectively. While the drop in delivery ratio is expected, the decrease is moderate—around 3% and 8% drop when errors occur for 10% or 30% of time. In fact, we ideally do not want to consider those fluctuating links when connecting nodes to form a backbone (if not necessary), and a more careful design of a backbone construction scheme that jointly considers link quality and node capacity is an interesting future research topic.

VI. Related Work

Many distributed algorithms have been proposed to find a connected dominating set. Das and Bharghavan [3] apply well-known centralized algorithms [6]. Using the unit-disk graph model, Wan et al. [4] propose a message-optimal algorithm that achieves a constant approximation ratio. Dubhashi et al. [5] propose a distributed algorithm that finds an $O(\log \Delta)$ approximation to the minimum connected dominating set in $O(\log n \log \Delta)$ time. None of them consider backbone maintenance or node capacity. As described in Section V, SPAN [7], GAF [8], and the scheme by Wu et al. [9] all consider remaining energy level when finding a connected backbone. However, none achieves a maximum-capacity backbone, nor provides a bound on the backbone size. Recently, Basagni et al. [20] present simulation results and compare a number of existing schemes for connected backbone construction in various aspects such as completion time, backbone size, and message overhead. Also related are clustering algorithms for sensor networks [27, 28]. Kuhn et al. [27] propose a clustering algorithm that finds a dominating set in the initial deployment phase. HEED [28] selects cluster-heads based on the residual energy and parameters such as node degree, but assumes that the network is quasi-stationary. In contrast, TRUNC-K provides backbone-maintenance mechanisms for dynamic networks. Recently, Lee et al. [29] present a backbone-construction scheme in a selfish environment and report experimental results from an implementation on a 12-node testbed.

TRUNC-K uses the sleep mode to save energy. There are schemes that exploit the sleep mode operation, but are not based on the connected backbone approach. A number of wireless MAC protocols attempt to save energy by putting nodes to sleep mode depending on various criteria [30, 31]. Zheng and Kravets [32] propose an on-demand power saving scheme, where nodes stay awake according to traffic load and their soft-state timers. Assuming dense sensor networks, ASCENT [33] uses a simple mechanism to determine "active" and "passive" nodes. However, ASCENT does not guarantee network connectivity. We also can achieve energy saving through transmission power control at each node [34–36], which is complementary to our work.

VII. Conclusions and Future Work

We have presented a parameterized scheme TRUNC-K that builds a connected backbone in multihop wireless networks. We have also proved that our scheme can construct essentially best possible backbones with respect to backbone size and node capacity. We generalized our scheme to construct and maintain a resilient backbone in dynamic networks. Through detailed simulations, we demonstrated that our proposed scheme outperforms existing energy-saving techniques in many aspects. In the future, we plan to investigate how to adjust the K value according to network environments (e.g., node mobility or density). Then, we will be able to include adaptive protocol mechanisms that can automatically change the K value when network parameters change over time (e.g., increased mobility, or new node deployment). We also want to analytically investigate the backbone performance with different K values. As discussed earlier, another interesting future direction is to design a backbone construction scheme that jointly considers node capacity and link quality.

Acknowledgment. We thank the reviewers for their valuable feedback.

References

[1] D. Johnson and D. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic Publishers, 2001.

[2] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), October 2003. IETF RFC 3626.

[3] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Proc. of ICC*, June 1997.

[4] P. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, 2002.

[5] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. *Journal of Computer and System Sciences*, 71, 2005.

[6] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms*, 1996.

[7] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5), 2002.

[8] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proc. of MobiCom*, 2001.

[9] J. Wu, M. Gao, and I. Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *Proc. of IEEE ICPP*, 2001.

[10] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM*, 2004.

[11] M. Gerla and J. T. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[12] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Symposium on computational geometry*, 2001.

[13] B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.

[14] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[15] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4), 2002.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*. MIT press, 2001.

[17] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Languages and Systems*, 1983.

[18] R. E. Tarjan. *Data structures and network algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1983.

[19] Fei Dai and Jie Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):908–920, 2004.

[20] S. Basagni, M. Mastrogiovanni, A. Panconesi, and C. Petrioli. Localized protocols for ad hoc clustering and backbone formation: A performance comparison. *IEEE Transactions on Parallel and Distributed Systems*, 17(4), 2006.

[21] The VINT Project. The Network Simulator–ns-2. Available at http://www.isi.edu/nsnam/ns.

[22] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE 802.11 Standard, 1999.

[23] J. Yoon, M. Liu, and B. D. Noble. Random waypoint considered harmful. In *Proceedings of Infocom*, April 2003.

[24] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of Mobicom*. ACM Press, 2002.

[25] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. of IEEE INFOCOM*, 2001.

[26] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of Mobicom*, pages 134–146. ACM Press, 2003.

[27] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proceedings of ACM MobiCom*, 2004.

[28] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *IEEE INFOCOM*, 2004.

[29] S. Lee, D. Levin, V. Gopalakrishnan, and B. Bhattacharjee. Backbone construction in selfish wireless networks. In *ACM Sigmetrics*, June 2007.

[30] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE Infocom*. IEEE, June 2002.

[31] Suresh Singh and C.S. Raghavendra. PAMAS: Power Aware Multi-Access Protocol with Signalling for Ad Hoc Networks. *SIGCOMM Computer Communication Review*, 28(3):5–26, July 1998.

[32] R. Zheng and R. Kravets. On-demand power management for ad hoc networks. In *Proc. of IEEE Infocom*, April 2003.

[33] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. *IEEE Trans. on Mobile Computing*, 3(3), 2004.

[34] V. Rodoplu and T. H. Meng. Minimum energy mobile wireless networks. *IEEE JSAC*, 17(8), August 1999.

[35] L. Li, J. Y. Halpern, P. Bahl, Y. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceedings of ACM symposium PODC*, 2001.

[36] N. Li and J. C. Hou. FLSS: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of Mobicom*, 2004.

[37] C. M. Fortuin, F. Ginibre, and P. N. Kasteleyn. Correlational inequalities for partially ordered sets. *Communications of Mathematical Physics*, 1971.

[38] M. Hofri. *Analysis of Algorithms*. Oxford University Press, 1995.

[39] H. Shachnai and A. Srinivasan. Finding large independent sets in graphs and hypergraphs. *SIAM Journal on Discrete Mathematics*, 18:488–500, 2004.

APPENDIX

We describe a special case of the FKG inequality [37]. Consider an event F that is determined by a vector $\vec{Y} = (Y_1, \cdots, Y_m)$ of independent random variables $Y_i \in \{0, 1\}$. Suppose that whenever F holds for $\vec{a}$, F also holds for any $\vec{b}$ that coordinate-wise dominates $\vec{a}$ (i.e., $\forall i, a_i \leq b_i$). Then, we call F an *increasing* event of $\vec{Y}$. For increasing events $F_1, \cdots, F_l$, the following holds:

$$Pr(\textstyle\bigwedge_{i=1}^{l} F_i) \quad \geq \quad \prod_{i=1}^{l} Pr(F_i). \tag{3}$$

We also use the following identity from [38], which holds for any non-negative integer d and any real $x \notin \{-d, -d+1, \ldots, 0\}$:

$$\sum_{l=0}^{d} \binom{d}{l} \frac{(-1)^l}{x+l} = \frac{1}{x\binom{d+x}{d}} \tag{4}$$

A. Proof of Theorem 2.2

We prove that in any D-regular graph with n nodes,

$$E[|\mathcal{L}|] \leq c\frac{n}{D}\log(D+1) \tag{5}$$

for a parameter c that can be arbitrarily close to 1 as D becomes large. Consider an indicator variable X_v, where $X_v = 1$ iff $v \in \mathcal{L}$. Then, $|\mathcal{L}| = \sum_v X_v$. Let us denote by P_v the probability that node v is nominated as leader. Then, from the linearity of expectation, $E[|\mathcal{L}|] = E[\sum_v X_v] = \sum_v E[X_v] = \sum_v P_v$. Then, to prove the theorem, it is sufficient to show:

$$\forall\, v, P_v \leq \frac{c}{D}\log(D+1). \tag{6}$$

Since all nodes have exactly D neighbors, P_v is same for all v's. To show the above inequality, we use the following: $P_v = 1 - Pr[\text{no nodes nominate } v] = 1 - \int_0^1 Pr[\text{no nodes nominate } v | c_v = t]\, dt$.

We define $\mathcal{E}_i = Pr[i\text{-th neighbor of } v \text{ does not nominate } v]$. We also denote $\mathcal{E}_0 = Pr[v \text{ itself nominates other node}]$. For each node u, consider the following random variable: $Y_u = 1$ if $c_u > c_v$; $Y_u = 0$ otherwise. Then, $\mathcal{E}_i\, (0 \leq i \leq D)$ is an increasing event of n random variable Y_u's, and we can apply the FKG inequality to P_v, similarly to [39]:

$$\begin{aligned} P_v &= 1 - \textstyle\int_0^1 Pr[\mathcal{E}_0 \wedge \mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_D | c_v = t]\, dt \\ &\leq 1 - \textstyle\int_0^1 \prod_{i=0}^{D} Pr[\mathcal{E}_i | c_v = t]\, dt \\ &= 1 - \textstyle\int_0^1 (1-t^D)^{D+1}\, dt \end{aligned}$$

Let us define $A = \int_0^1 (1-t^D)^{D+1}\, dt$. Then, $P_v \leq 1 - A$. We want to find a value $c \geq 1$ that satisfies the following:

$$A \quad \geq \quad (\tfrac{1}{D+1})^{\frac{c}{D}} \tag{7}$$

Then, since $x \geq 1 + \log x$ for all $x > 0$, we have $A \geq 1 - c\log(D+1)/D$, and consequently, $P_v \leq c\log(D+1)/D$, which is what we want to show.

Now, it remains to determine c. By taking the natural logarithm of (7), c should satisfy:

$$c \quad \geq \quad -\tfrac{D}{\log(D+1)}\log A = \tfrac{D}{\log(D+1)}\log(A^{-1}). \tag{8}$$

We further simplify A by using (4):

$$\begin{aligned} A &= \sum_{l=0}^{D+1} (-1)^l \binom{D+1}{l} \int_0^1 t^{Dl}\, dt \\ &= \sum_{l=0}^{D+1} \frac{(-1)^l}{Dl+1}\binom{D+1}{l} = \binom{D+1+1/D}{D+1}^{-1} \\ &= \left(\textstyle\prod_{i=1}^{D+1}(1+\frac{1}{Di})\right)^{-1} \end{aligned} \tag{9}$$

We note the following facts: $H(n) = \sum_{i=1}^{n} 1/i \leq \log n + 1$, and $1 + x \leq e^x$ for all $x \geq 0$. We denote $\exp(x) = e^x$. So, we can upper-bound the right-hand-side of (8) as:

$$\begin{aligned} \tfrac{D}{\log(D+1)}\log(A^{-1}) &= \tfrac{D}{\log(D+1)}\log\left(\textstyle\prod_{i=1}^{D+1}(1+\frac{1}{Di})\right) \\ &\leq \tfrac{D}{\log(D+1)}\log\left(\textstyle\prod_{i=1}^{D+1}\exp(\frac{1}{Di})\right) \\ &= \tfrac{D}{\log(D+1)}\textstyle\sum_{i=1}^{D+1}\frac{1}{Di} \\ &\leq \tfrac{\log(D+1)+1}{\log(D+1)} \end{aligned}$$

Consequently, using:

$$c = 1 + \tfrac{1}{\log(D+1)} \tag{10}$$

satisfies (8). Note that as D grows large, c approaches 1.

B. Proof of Lemma 2.3

Let us denote c from (10) as $c(D)$. Note that $c(D)$ is decreasing where $c(1) = 1 + 1/\log 2 \approx 2.44$. Using similar steps as in Appendix A, we can show:

$$E[|\mathcal{L}|] \leq \textstyle\sum_{v\in V} \alpha\, c(d_v)\frac{\log(d_v+1)}{d_v} \leq c' \sum_{v\in V}\frac{\log(d_v+1)}{d_v},$$

where $c' = \alpha\, c(\delta)$ and δ is the minimum degree in the network.

C. Proof of Theorem 2.4

We first show that in an α-locally-regular graph, the size of an optimal dominating set (*OPT*) is lower-bounded by $OPT \geq c'' \sum_{v \in V} \frac{1}{d_v}$, for some constant c''. To obtain this bound, we first consider the following Integer Program (IP) for a minimum dominating set:

$$\begin{array}{ll} \text{minimize} & \sum_{v \in V} x_v, \\ \text{subject to} & \forall v,\ x_v + \sum_{u:(u,v) \in E} x_u \geq 1, \quad x_v \in \{0,1\} \end{array}$$

Relaxing the integrality constraints, we get a Linear Program (LP) where $\forall v, x_v \geq 0$. An optimal LP solution is a lower bound on the optimal solution of IP. The dual of the LP is:

$$\begin{array}{ll} \text{maximize} & \sum_{v \in V} x_v, \\ \text{subject to} & \forall v,\ x_v + \sum_{u:(u,v) \in E} x_u \leq 1,\ x_v \geq 0 \end{array}$$

We show that for $c'' = \frac{1}{\alpha+1}$, using $x_v = \frac{c''}{d_v}$ is a feasible solution of the dual. Clearly, x_v is positive. We now show that this choice of x_v meets the first set of constraints as well. Note that in α-locally-regular graphs, $d_u \geq d_v/\alpha$.

$$\begin{array}{rcl} x_v + \sum_{u:(u,v) \in E} x_u & = & \frac{c''}{d_v} + \sum_{u:(u,v) \in E} \frac{c''}{d_u} \\ & \leq & c'' + \sum_{u:(u,v) \in E} \frac{c''}{d_v/\alpha} \\ & = & c'' + \frac{c''\alpha}{d_v} d_v \ = \ c''(1+\alpha) = 1. \end{array}$$

Since any feasible solution for the dual is a lower bound on *OPT*, $OPT \geq c'' \sum_{v \in V} \frac{1}{d_v}$. From Lemma 2.3,

$$E[|\mathcal{L}|] \leq \ c' \sum_{v \in V} \frac{1}{d_v} \log(\Delta + 1) \ \leq \ \frac{c'}{c''} \log(\Delta + 1)\, OPT.$$

D. Proof of Theorem 3.1

We denote by v the minimum-capacity node of the resulting backbone. If $v \in \mathcal{L}$, by Theorem 2.1, the backbone is a maximum-capacity connected dominating set. We study the case where v is a part of a virtual edge. Consider the virtual edge that included v while connecting fragments F_1 and F_2. Pick any two leaders each from F_1 and F_2 and call them L_1 and L_2, respectively. We first prove by contradiction that the following set is not connected: $S = \{u \mid c_u > c_v\}$. Note that $L_1 \in S$ and $L_2 \in S$. Let us assume S is connected. Then, L_1 and L_2 have at least one path P consisting only of nodes in S. In this case, F_1 and F_2 can get merged using possibly multiple virtual edges using the nodes on P. As a result, F_1 and F_2 would not have chosen the virtual edge with v. This contradiction proves that S is not connected. Since S is not connected, no subset of S can be a connected dominating set. Therefore, any connected dominating set must include a node whose capacity is at most c_v.

E. Proof of the $(2H(\Delta)+1)$-bound in [3]

The proof for the the $(2H(\Delta) + 1)$ bound in Lemma 2 in [3] is not available in the literature, and we present the proof here. Their algorithm first finds a dominating set *DS* using the greedy heuristic [16], which guarantees $s \doteq |DS| \leq H(\Delta)OPT$. Then, when we connect fragments using an MST-based algorithm with chains of up to two nodes, we should first connect two components using one node whenever available. If no pairs of components have common neighbors, then we connect components using chains of two nodes. Suppose that we have added x nodes before using chains of two nodes. Let us denote by y the number of remaining components at this point. Clearly, $y \leq s - x$. Since these components have no common neighbors, each component requires at least one distinct node in *OPT*, and $y \leq OPT$. We now use 2 nodes for each connection of remaining components, giving a total cost of: $s + x + 2(y-1) \leq 2s + y \leq (2H(\Delta)+1)OPT$.

Seungjoon Lee received his Bachelor's and Master's degrees in Computer Science from Seoul National University, Seoul, Korea, in 1996 and 2000 and his Ph. D. in Computer Science from the University of Maryland in 2006. Currently, he is a senior member of technical staff in AT&T Labs, Research. His research interests include wireless networks, mobile computing, peer-to-peer systems, and multicasting.

Bobby Bhattacharjee received Bachelor's degrees in Computer Science and Mathematics from Georgia College in 1994 and his Ph. D. in Computer Science from the College of Computing at Georgia Tech. in 1999. He is currently an Associate Professor in the Department of Computer Science at the University of Maryland. Dr. Bhattacharjee's research interests are in the design and implementation of wide-area networking, distributed systems, and security protocols. His current focus is on the design of decentralized secure systems for multi-party applications especially in the context of peer-to-peer and overlay systems.

Aravind Srinivasan (Senior Member, IEEE) is a Professor at the University of Maryland. He received his degrees from Cornell University (Ph.D.) and the Indian Institute of Technology, Madras (B.Tech.). His research interests are in randomized algorithms, networking, social networks, combinatorial optimization, and related areas. He has published several papers in these areas, in journals including Nature, Journal of the ACM, IEEE/ACM Transactions on Networking, and SIAM Journal on Computing. He is an editor of four journals, and has served on the program committees of various conferences.

Samir Khuller received his M.S and Ph.D from Cornell University in 1989 and 1990, respectively. He spent 2 years as a Research Associate at the Institute for Advanced Computer Studies at the University of Maryland, before joining the Computer Science Department in 1992, where he is a Professor and Associate Chair in the Department of Computer Science.

His research interests are in graph algorithms, discrete optimization, and computational geometry. He has published about 130 journal and conference papers, and several book chapters on these topics. He received the National Science Foundation's Career Development Award, the Dean's Teaching Excellence Award and also a CTE-Lilly Teaching Fellowship. In 2003, he and his students were awarded the "Best newcomer paper" award for the ACM PODS Conference. He received the University of Maryland's Distinguished Scholar Teacher Award in 2007.